M&G No. 60001.0263US01/MS303917.1

PROGRAMMABLE OBJECT MODEL FOR EXTENSIBLE MARKUP LANGUAGE SCHEMA VALIDATION

Related Applications

United States Utility Patent Application by applicant matter number 60001.0264US01/MS303918.1, entitled "Programmable Object Model for Extensible Markup Language Markup in an Application," and United States Utility Patent Application by applicant matter number 60001.0270US01/MS303919.1, entitled "Programmable Object Model for Namespace or Schema Library Support in a Software Application," are hereby incorporated by reference.

Copyright Notice

Field of the Invention

The present invention relates generally to programmable object models. More particularly, the present invention relates to a programmable object model for Extensible Markup Language (XML) schema validation.

Background of the Invention

Computer software applications allow users to create a variety of documents to assist them in work, education, and leisure. For example, popular word processing applications allow users to create letters, articles, books, memoranda, and the like. Spreadsheet applications allow users to store, manipulate, print, and display a

variety of alphanumeric data. Such applications have a number of well known strengths including rich editing, formatting, printing, calculation and on-line and off-line editing.

Most computer software applications do not contain all necessary programming for providing functionality required or desired by every potential user. Many programmers often wish to take advantage of an existing application's capabilities in their own programs or to customize the functionality of an application and make it more suitable for a specific set of users or actions. For example, a programmer working in the financial industry may wish to customize a word processor for a user audience consisting of financial analysts editing financial reports. In recent years, the Extensible Markup Language has been used widely as an interchangeable data format for many users. A word processor, spreadsheet application or other application capable of validating Extensible Markup Language (XML) data against an attached or associated XML schema file provides value to its users. However, because the functionality of the schema validation model of the application is not exposed to the programmer, the programmer is not able to customize or modify the schema validation model to work with the programmer's own customized versions of the application or documents prepared by the application.

Accordingly, there is a need in the art for a programmable object model for allowing users/programmers to programmatically access, supplement or otherwise modify an application's XML validation functionality. It is with respect to these and other considerations that the present invention has been made.

Summary of the Invention

The present invention provides methods and systems for allowing a user to access programmatically the schema validation model of an application via a programmable object model. The programmable object model includes a plurality of object-oriented message calls or application programming interfaces for allowing a user to programmatically access the schema validation model of an application by sending message calls and required parameters to the schema validation model in order to customize or otherwise modify the operation of the schema validation model as desired

by the user. Once the user has access to the schema validation model of a given application, the user may attach Extensible Markup Language (XML) schema files to a document and control definitions, grammatical rules, and other settings dictated by the attached XML schema file. Accessing the schema validation model of a given application also allows the user to customize the application's native schema validation functionality with user-defined rules and error text. According to aspects of the invention, the XML schema validation model of a given application may be accessed by the user from a user-created program written according to a variety of different languages such as C, C++, C#, Visual Basic and the like.

These and other features, advantages, and aspects of the present invention may be more clearly understood and appreciated from a review of the following detailed description of the disclosed embodiments and by reference to the appended drawings and claims.

Brief Description of the Drawings

Fig. 1 is a simplified block diagram of a computing system and associated peripherals and network devices that provide an exemplary operating environment for the present invention.

Fig. 2 is a simplified block diagram illustrating interaction between software objects according to an object-oriented programming model.

Fig. 3 is a block diagram illustrating interaction between a document, an attached schema file, and a schema validation functionality model.

Fig. 4 is a block diagram illustrating interaction between a third party customized application, a third party customized schema validation model and a native schema validation model.

Detailed Description of the Preferred Embodiment

Embodiments of the present invention are directed to methods and systems for allowing a user to programmatically call a native schema validation model of a software application for customizing, utilizing, and otherwise manipulating objects

and properties of the native schema validation model for allowing the user to attach one or more desired XML schemas to a document and for controlling one or more objects or properties of those attached XML schema files as applied to a document to which the one or more schema files are attached. Embodiments of the present invention also allow a user to programmatically customize a native schema validation model with user-defined rules and error text. In the following detailed description, references are made to the accompanying drawings that forma part hereof, and in which are shown by way of illustrations specific embodiments or examples. These embodiments may be combined, other embodiments may be utilized, and structural changes may be made without departing from the spirit or scope of the present invention. The following detailed description is therefore not to be taken in a limiting senses and the scope of the present invention is defined by the appended claims and their equivalents.

Referring now to the drawings, in which like numerals represent like elements through the several figures, aspects of the present invention and the exemplary operating environment will be described. Fig. 1 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. While the invention will be described in the general context of program modules that execute in conjunction with an application program that runs on an operating system on a personal computer, those skilled in the art will recognize that the invention may also be implemented in combination with other program modules.

Generally, program modules include routines, programs, components, data structures, and other types of structures that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications

network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

Turning now to Fig. 1, an illustrative computer architecture for a personal computer 2 for practicing the various embodiments of the invention will be described. The computer architecture shown in Fig. 1 illustrates a conventional personal computer, including a central processing unit 4 ("CPU"), a system memory 6, including a random access memory 8 ("RAM") and a read-only memory ("ROM") 10, and a system bus 12 that couples the memory to the CPU 4. A basic input/output system containing the basic routines that help to transfer information between elements within the computer, such as during startup, is stored in the ROM 10. The personal computer 2 further includes a mass storage device 14 for storing an operating system 16, application programs, such as the application program 305, and data.

The mass storage device 14 is connected to the CPU 4 through a mass storage controller (not shown) connected to the bus 12. The mass storage device 14 and its associated computer-readable media, provide non-volatile storage for the personal computer 2. Although the description of computer-readable media contained herein refers to a mass storage device, such as a hard disk or CD-ROM drive, it should be appreciated by those skilled in the art that computer-readable media can be any available media that can be accessed by the personal computer 2.

By way of example, and not limitation, computer-readable media may comprise computer storage media and communication media. Computer storage media includes volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EPROM, EEPROM, flash memory or other solid state memory technology, CD-ROM, DVD, or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer.

According to various embodiments of the invention, the personal computer 2 may operate in a networked environment using logical connections to remote computers through a TCP/IP network 18, such as the Internet. The personal computer 2 may connect to the TCP/IP network 18 through a network interface unit 20 connected to the bus 12. It should be appreciated that the network interface unit 20 may also be utilized to connect to other types of networks and remote computer systems. The personal computer 2 may also include an input/output controller 22 for receiving and processing input from a number of devices, including a keyboard or mouse (not shown). Similarly, an input/output controller 22 may provide output to a display screen, a printer, or other type of output device.

As mentioned briefly above, a number of program modules and data files may be stored in the mass storage device 14 and RAM 8 of the personal computer 2, including an operating system 16 suitable for controlling the operation of a networked personal computer, such as the WINDOWS XP operating system from MICROSOFT CORPORATION of Redmond, Washington. The mass storage device 14 and RAM 8 may also store one or more application programs. In particular, the mass storage device 14 and RAM 8 may store an application program 305 for creating and editing an electronic document 310. For instance, the application program 305 may comprise a word processing application program, a spreadsheet application, a contact application, and the like. Application programs for creating and editing other types of electronic documents may also be used with the various embodiments of the present invention. A schema file 330and a namespace/schema library 400, described below, are also shown.

Exemplary embodiments of the present invention are implemented by communications between different software objects in an object-oriented programming environment. For purposes of the following description of embodiments of the present invention, it is useful to briefly to describe components of an object-oriented programming environment. Fig. 2 is a simplified block diagram illustrating interaction between software objects according to an object-oriented programming model. According to an object-oriented programming environment, a first object 210 may include software code, executable methods, properties, and parameters. Similarly, a

second object 220 may also include software code, executable methods, properties, and parameters.

A first object 210 may communicate with a second object 220 to obtain information or functionality from the second object 220 by calling the second object 220 via a message call 230. As is well know to those skilled in the art of object-oriented programming environment, the first object 210 may communicate with the second object 220 via application programming interfaces (API) that allow two disparate software objects 210, 220 to communicate with each other in order to obtain information and functionality from each other. For example, if the first object 210 requires the functionality provided by a method contained in the second object 220, the first object 210 may pass a message call 230 to the second object 220 in which the first object identifies the required method and in which the first object passes any required parameters to the second object required by the second object for operating the identified method. Once the second object 220 receives the call from the first object, the second object executes the called method based on the provided parameters and sends a return message 250 containing a value obtained from the executed method back to the first object 210.

For example, in terms of embodiments of the present invention, and as will be described below, a first object 210 may be a third party customized application that passes a message to a second object such as an Extensible Markup Language schema validation object whereby the first object identifies a method requiring the validation of a specified XML element in a document where the specified XML element is a parameter passed by the first object with the identified method. Upon receipt of the call from the first object, according to this example, the schema validation object executes the identified method on the specified XML element and returns a message to the first object in the form of a result or value associated with the validated XML element. Operation of object-oriented programming environments, as briefly described above, are well known to those skilled in the art.

As described below, embodiments of the present invention are implemented through the interaction of software objects in the use, customization, and

application of components of the Extensible Markup Language (XML). Fig. 3 is a block diagram illustrating interaction between a document, an attached schema file, and a schema validation functionality module. As is well known to those skilled in the art, the Extensible Markup Language (XML) provides a method of describing text and data in a document by allowing a user to create tag names that are applied to text or data in a document that in turn define the text or data to which associated tags are applied. For example referring to Fig. 3, the document 310 created with the application 305 contains text that has been marked up with XML tags 315, 320, 325. For example, the text "Greetings" is annotated with the XML tag <title>. The text "My name is Sarah" is annotated with the <body> tag. According to XML, the creator of the <title> and <body> tags is free to create her own tags for describing the tags to which those tags will be applied. Then, so long as any downstream consuming application or computing machine is provided instructions as to the definition of the tags applied to the text, that application or computing machine may utilize the data in accordance with the tags. For example, if a downstream application has been programmed to extract text defined as titles of articles or publications processed by that application, the application may parse the document 310 and extract the text "Greetings," as illustrated in Fig. 3 because that text is annotated with the tag <title>. The creator of the particular XML tag naming for the document 310, illustrated in Fig. 3, provides useful description for text or data contained in the document 310 that may be utilized by third parties so long as those third parties are provided with the definitions associated with tags applied to the text or data.

According to embodiments of the present invention, the text and XML markup entered into the document 310 may be saved according to a variety of different file formats and according to the native programming language of the application 305 with which the document 310 is created. For example, the text and XML markup may be saved according to a word processing application, a spreadsheet application, and the like. Alternatively, the text and XML markup entered into the document 310 may be saved as an XML format whereby the text or data, any applied XML markup, and any formatting such as font, style, paragraph structure, etc. may be saved as an XML

representation. Accordingly, downstream or third party applications capable of understanding data saved as XML may open and consume the text or data thus saved as an XML representation. For a detailed discussion of saving text and XML markup and associated formatting and other attributes of a document 310 as XML, see U.S. Patent Application entitled "Word Processing Document Stored in a Single XML File that may be Manipulated by Applications that Understanding XML," U.S. Serial No. 10/187,060, filed June 28, 2002, which is incorporated herein by reference as if fully set out herein.

In order to provide a definitional framework for XML markup elements (tags) applied to text or data, as illustrated in Fig. 3, XML schema files are created which contain information necessary for allowing users and consumers of marked up and stored data to understand the XML tagging definitions designed by the creator of the document. Each schema file also referred to in the art as a Namespace or XSD file preferably includes a listing of all XML elements (tags) that may be applied to a document according to a given schema file. For example, a schema file 330, illustrated in Fig. 3, may be a schema file containing definitions of certain XML elements that may be applied to a document 310 including attributes of XML elements or limitations and/or rules associated with text or data that may be annotated with XML elements according to the schema file. For example, referring to the schema file 330 illustrated in Fig. 3, the schema file is identified by the Namespace "intro" the schema file includes a root element of <intro card>.

According to the schema file 330, the <intro card> element serves as a root element for the schema file and also as a parent element to two child elements <title> and <body>. As is well known to those skilled in the art, a number of parent elements may be defined under a single root element, and a number of child elements may be defined under each parent element. Typically, however, a given schema file 330 contains only one root element. Referring still to Fig. 3, the schema file 330 also contains attributes 340 and 345 to the <title> and <body> elements, respectfully. The attributes 340 and 345 may provide further definition or rules associated with applying the respective elements to text or data in the document 310. For example, the attribute 345 defines that text annotated with the <title> element must be less than or equal to

twenty-five characters in length. Accordingly, if text exceeding twenty-five characters in length is annotated with the <title> element or tag, the attempted annotation of that text will be invalid according to the definitions contained in the schema file 330.

By applying such definitions or rules as attributes to XML elements, the creator of the schema may dictate the structure of data contained in a document associated with a given schema file. For example, if the creator of a schema file 330 for defining XML markup applied to a resume document desires that the experience section of the resume document contain no more than four present or previous job entries, the creator of the schema file 330 may define an attribute of an <experience> element, for example, to allow that no more than four present or past job entries may be entered between the <experience> tags in order for the experience text to be valid according to the schema file 330. As is well known to those skilled in the art, the schema file 330 may be attached to or otherwise associated with a given document 310 for application of allowable XML markup defined in the attached schema file to the document 310. According to one embodiment, the document 310 marked up with XML elements of the attached or associated schema file 330 may point to the attached or associated schema file by pointing to a uniform resource identifier (URI) associated with a Namespace identifying the attached or associated schema file 330.

According to embodiments of the present invention, a document 310 may have a plurality of attached schema files. That is, a creator of the document 310 may associate or attach more than one schema file 330 to the document 310 in order to provide a framework for the annotation of XML markup from more than one schema file. For example, a document 310 may contain text or data associated with financial data. A creator of the document 310 may wish to associate XML schema files 330 containing XML markup and definitions associated with multiple financial institutions. Accordingly, the creator of the document 310 may associate an XML schema file 330 from one or more financial institutions with the document 310. Likewise, a given XML schema file 330 may be associated with a particular document structure such as a template for placing financial data into a desirable format.

According to embodiments of the present invention, a collection of XML schema files and associated document solutions may be maintained in a Namespace or schema library located separately from the document 310. The document 310 may in turn contain pointers to URIs in the Namespace or schema library associated with the one or more schema files attached to otherwise associated with the document 310. As the document 310 requires information from one or more associated schema files, the document 310 points to the Namespace or schema library to obtain the required schema definitions. For a detailed description of the use of an operation of Namespace or schema libraries, see U.S. Patent Application entitled "System and Method for Providing Namespace Related Information," U.S. Serial No. 10/184,190, filed June 27, 2002, and U.S. Patent Application entitled "System and Method for Obtaining and Using Namespace Related Information for Opening XML Documents," U.S. Serial No. 10/185,940, filed June 27, 2002, both U.S. patent applications of which are incorporated herein by reference as if fully set out herein. For a detailed description of a mechanism for downloading software components such as XML schema files and associated solutions from a Namespace or schema library, see US Patent Application entitled Mechanism for Downloading Software Components from a Remote Source for Use by a Local Software Application, US Serial No. 10/164,260, filed June 5, 2002.

Referring still to Fig. 3, a schema validation functionality module 350 is illustrated for validating XML markup applied to a document 310 against an XML schema file 330 attached to or otherwise associated with the document 310, as described above. As described above, the schema file 330 sets out acceptable XML elements and associated attributes and defines rules for the valid annotation of the document 310 with XML markup from an associated schema file 330. For example, as shown in the schema file 330, two child elements <title> and <body> are defined under the root or parent element <intro card>. Attributes 340, 345 defining the acceptable string length of text associated with the child elements <title> and <body> are also illustrated. As described above, if a user attempts to annotate the document 310 with XML markup from a schema file 330 attached to or associated with the document in violation of the XML markup definitions contained in the schema file 330, an invalidity or error state

will be presented. For example, if the user attempts to enter a title string exceeding twenty-five characters, that text entry will violate the maximum character length attribute of the <title> element of the schema file 330. In order to validate XML markup applied to a document 310, against an associated schema file 330, a schema validation module 350 is utilized. As should be understood by those skilled in the art, the schema validation module 350 is a software module including computer executable instructions sufficient for comparing XML markup and associated text entered in to a document 310 against an associated or attached XML schema file 330 as the XML markup and associated text is entered in to the document 310.

According to embodiments of the present invention, the schema validation module 350 compares each XML markup element and associated text or data applied to the document 310 against the attached or associated schema file 330 to determine whether each element and associated text or data complies with the rules and definitions set out by the attached schema file 330. For example, if a user attempts to enter a character string exceeding twenty-five characters annotated by the <title> elements 320, the schema validation module will compare that text string against the text string attribute 340 of the attached schema file 330 and determine that the text string entered by the user exceeds the maximum allowable text string length. Accordingly, an error message or dialogue will be presented to the user to alert the user that the text string being entered by the user exceeds the maximum allowable character length according to the attached schema file 330. Likewise, if the user attempts to add an XML markup element between the <title> and the <body> elements, the schema validation module 350 will determine that the XML markup element applied by the user is not a valid element allowed between the <title> and <body> elements according to the attached schema file 330. Accordingly, the schema validation module 350 will generate an error message or dialogue to the user to alert the user of the invalid XML markup.

Programmable Object Model for XML Schema Validation

Fig. 4 is a block diagram illustrating interaction between a third party customized application, a third party customized schema validation model and a native schema validation model. According to embodiments of the present invention, an object-oriented programming model is provided for exposing to a user the native XML schema validation functionality of a given software application 305 to allow the user to programmatically call the schema validation functionality via a set of application programming interfaces or object-oriented message calls either directly through one or more application programming interfaces or programmatically through other software application programs written according to a variety of programming languages, for example C, C++, C#, Visual Basic, and the like.

With access to the native schema validation functionality of an application 305, the user may manually or programmatically attach one or more XML schema files 330 to a document 310 and subsequently control components of those XML schema files including individual components, objects or properties thereof. Additionally, access to the schema validation functionality of a given application 305 allows a user to customize the application's native schema validation functionality with user-defined rules and error text. For example, as described above with reference to Fig. 3, if according to a given XML schema file 330 an error is produced if a user attempts to add a text string in excess of that allowed by the schema definitions for a given XML markup element, an error message may be presented to the user upon validation of the user's text entry against the schema file 330. According to embodiments of the present invention, by allowing the user access to the native schema validation functionality of the application 305 with which the document 310 is created, the user may customize the schema validation functionality to allow the text string in excess of the maximum character length, or the user may customize the validation functionality to present a personalized or customized error message in place of the error message otherwise provided by the native schema validation model 410 of the application 305 with which the document 310 is created.

Referring to Fig. 4, according to embodiments of the present invention, a user obtains access to the native schema validation model 410 via one or more application programming interfaces or message calls passed from a third party customized application 430 to the native schema validation model 410, or passed from a third party customized validation model 420 to the native schema validation model 410. In either case, as described above with respect to Fig. 2, according to object-oriented programming environments, a message call passed from the third party customized application 430 or the third party customized validation model 420 to the native schema validation model 410 of a given software application allows the third party to pass, as message parameters, XML schema file attachment information or schema validation model modifications for receipt by the native schema validation model. In return, the native schema validation model may return a value to the third party custom application 430 or to the third party customized validation model 420 to allow the third party user to make modifications to one or more schema files attached to a document 310, or to allow the user to customize the application's native schema validation model 410 to customize the validation rules associated with the native schema validation model 410. For example, if the native schema validation model 410 presents an error to the user upon the entry of mixed XML content into a document 310 where the mixed content includes entry of data into the document 310 not associated with an XML markup element from the attached XML schema file 330, a third party may utilize the application programming interfaces or message calls according to the present invention to modify the native schema validation model 410 to allow mixed content data entry into the document 310 without receiving a validation error by the native schema validation model 410. Other exemplary uses and benefits thereof of exposing the native schema validation model to a user as described herein will be understood from the description that follows.

The following is a description of objects and associated properties comprising application programming interfaces (API) or object-oriented message calls that provide a user direct access to the native schema validation model 410 of an application 305 with which the user is editing a document 310. Following each of the

objects or associated properties set out below is a description of the operation and functionality of the object or associated property.

Application object - an object representing the application. The following are properties and methods of this object related to accessing an application's native schema validation model or object.

.XMLValidationError event

An event firing whenever a schema violation is detected by the application in the document containing XML markup. This event firing can pass the following parameters to an event handler procedure.

XMLNode – a pointer to the XML node where the violation has been detected.

Document object - an object representing the document. The following are properties and methods of this object related to accessing an application's native schema validation model or object.

.ChildNodeSuggestions property

A property pointing to a collection of XMLChildNodeSuggestions consisting of all the elements suggested by the application as available for insertion into the current document context based on the schema and the current context.

.XMLSchemaReferences property

A property pointing to the collection of XMLSchemaReferences representing all the schemas associated with this document.

.XMLSchemaViolations property

A property pointing to the XMLNodes collection consisting of all the XML nodes in a document that have a schema violation.

.XMLShowAdvancedErrors property

A property setting determining whether an application shows error messages related to schema validation in an advanced or simpler format.

XMLChildNodeSuggestions collection object - an object that is a collection of XMLChildNodeSuggestion objects representing XML elements suggested to the user by the application based on the XML schema and the current editing context. The following are properties and methods of this object related to accessing an application's native schema validation model or object.

.Application property

A read only pointer to the Application object representing the application of this object model.

.Count property

A read only property returning the number of XML schemas in the collection.

.Creator property

A read only pointer to the creator of the object.

.Item() method

A method for accessing the individual members of this collection using a numerical index or a search keyword. It can accept the following parameters.

Index – a number representing the position of the requested XMLChildNodeSuggestion object in the collection. The index can also be a text string representing the name of the requested node.

.Parent property

A read only property returning the parent object of the collection. This property returns a pointer to the document object that the XMLChildNodeSuggestions from which the collection is accessed.

XMLChildNodeSuggestion object - an object representing an XML element suggested by the application as available for insertion in the current editing context based on the XML schema and the current editing context. The following are properties and methods of this object related to accessing an application's native schema validation model or object.

.Application property

A read only pointer to the Application object representing the application of this object model.

.BaseName property

A property returning the name of a specified XML element without any prefixes.

.Creator property

A read only pointer to the creator of the object.

.Insert() method

A method for inserting an XML element suggested into the document. It returns the newly created XML node object. It can accept the following parameters.

Range – pointing to the part of the document where the suggested XML element is to be applied.

.NamespaceURI property

A property returning the Namespace of the suggested element.

.Parent property

A read only property returning the parent object of this object. This property returns a pointer to the collection of which the XMLChildNodeSuggestion object is a member.

.XMLSchemaReference property

A property pointing to the XMLSchemaReference object representing the schema file containing the suggested XML element.

XMLNode object - an object representing an XML node in the document. The following are properties and methods of this object related to accessing an application's native schema validation model or object.

.ChildNodeSuggestions property

A property pointing to an XMLChildNodeSuggestions collection consisting of XML elements suggested by the application as possible child nodes of this node based on the XML schema.

.SetValidationError() method

A method to override validation error notifications from the application's built-in validation model with a programmer's own custom errors. It can also be used to hide schema violations on an individual node basis as well as to enable the programmer to cause schema violations on otherwise valid nodes. It can accept the following parameters.

Status – determined whether a schema violation should be reported on the node or not.

ErrorText – determines the text to show as the schema violation description.

ClearedAutomatically – a flag indicating whether this schema violation will be cleared as soon as the application regains control or not.

.Validate() method

A method for on-demand validation of specific XML nodes in the document.

.ValidationErrorText property

A property returning the description text of the current schema violation on this node.

.ValidationStatus

A property indicating the validation status of the node, and distinguishing between valid and any number of invalid states.

XMLSchemaReferences collection object – an object representing a collection of schemas attached to a document. The following are properties and methods of this object related to accessing an application's native schema validation model or object.

.Add() method

A method to attach a new schema to a document. This method creates, inserts into the collection and returns a new XMLSchemaReference object representing the new schema[w1]. The parameters of this method are as follows. ([NamespaceURI], [Alias], [FileName], [InstallForAllUsers As Boolean = False] The Namespace URI names the namespace and the alias provides a shortened or fanciful name as desired by the developer. The file name includes a file name to which the schema is to be added.

.AllowSaveAsXMLWithoutValidation property

A property for controlling whether the application allows the user to save the document as XML even if it violates the schema.

.Application property

A read only pointer to the application object representing the application of this object model.

.AutomaticValidation property

A property controlling whether the application's real-time schema validation is enabled or not.

.Count property

A read only property returning the number of XML schemas attached to the document which is equivalent to the number of XMLSchemaReference objects in the collection.

.Creator property

A read only pointer to the creator of the object.

.IgnoreMixedContent property

A property that controls whether or not text nodes that have element siblings should be validated. When turned on, this property also prevents those text nodes from being saved into the XML format if the user has chosen to save the non-native XML markup only. This is intended to help prevent text that is inserted into the document by XSLT transformations from being treated as part of the data.

.Item() method

A method for accessing the individual members of this collection using an numerical index or a search keyword. It can accept the following parameters.

Index – a number representing the position of the requested XMLSchemaReference object in the collection. The index can also be a text string representing the Namespace of the requested schema.

.Parent property

A read only property returning the parent object of the collection. This property returns a pointer to the document object that the XMLSchemaReferences from which the collection is accessed.

.UnderlineValidationErrors property

A property controlling whether schema violations are highlighted in the document in any way, such as by being underlined.

.Validate() method

A method to validate the document against all the attached schemas on demand.

XMLSchemaReference object - an object representing an XML schema attached to the document. The following are properties and methods of this object related to accessing an application's native schema validation model or object.

.Application property

A read only pointer to the application object representing the application of this object model.

.Creator property

A read only pointer to the creator of the object.

.Delete()

A method for removing a schema reference from the document and the corresponding object from the XMLSchemaReferences collection.

.Location property

A property returning the location of the schema file.

.NamespaceURI property

A property returning the target namespace of the schema file.

.Parent property

A read only property returning the parent object of this object. This property returns a pointer to the collection from which the XMLSchemaReference object is a member.

.Reload() method

A method to reload this particular schema from the file. It is useful when the schema changes outside of the application and now the application needs to get an updated schema definition.

As described herein methods and systems are provided for allowing a user to programmatically call a schema validation model via a set of application programming interfaces or object-oriented message calls whereby the user may call for the execution of executable methods and pass desired properties and parameters to the schema validation model in order to customize the schema validation model or to change controls or settings utilized by the native schema validation model for validating XML markup applied to a document against a given XML schema file. It will be apparent to those skilled in the art that various modifications or variations may be made in the present invention without parting from the scope or spirit of the invention. Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein.